



Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./ Mrs. probin Bhag chandhari
of GCE-1 *Class,*

ID. No. 22 DCE 006 *has satisfactorily completed*

his/ her term work in CE-397 Mastering Competetive for
the ending in April 2024/2025 programming

Date : 1/2/25

Sign. of Faculty


Head of Department

Faculty of Technology and Engineering
Devang Patel Institute of Advance Technology and Research
Department of Computer Engineering

Practical List

Academic Year	:	2024-25	Semester	:	6 th
Course code	:	CE397	Course name	:	Competitive Programming 2

Sr. No.	Aim	Hrs	CO
1	Searching and sorting	2	4
1.1	<p>Ferris wheel: There are N children who want to go to a Ferris wheel in the form of an array arr[], and your task is to find a gondola for each child. Each gondola may have one or two children in it, and in addition, the total weight in a gondola may not exceed X. You know the weight of every child. What is the minimum number of gondolas needed for the children?</p> <p>Input: N = 4, X = 10, arr[] = {7, 2, 3, 9} Output: 3 Explanation: We need only 3 gondolas: {2, 3}, {7} and {9}.</p> <p>Input: N = 4, X = 6, arr[] = {2, 3, 3, 4} Output: 2 Explanation: We need only 2 gondolas: {2, 4} and {3, 3}</p>		
1.2	<p>Sort Vowels in a String Given a 0-indexed string s, permute s to get a new string t suchthat: All consonants remain in their original places. More formally,if there is an index i with 0 <= i < s.length such that s[i] is a consonant, then t[i] = s[i]. The vowels must be sorted in the nondecreasing order of their ASCII values. More formally, for pairs of indices i, j with 0 <=i < j < s.length such that s[i] and s[j] are vowels, then t[i] must not have a higher ASCII value than t[j]. Return the resulting string. The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in lowercase or uppercase. Consonants comprise all letters that arenot vowels.</p> <p>Example 1: Input: s = "lEetcOde"Output: "lEOtcede" Explanation: 'E', 'O', and 'e' are the vowels in s; 'l', 't', 'c', and 'd'are all consonants. The vowels are sorted according to their ASCII values, and the consonants remain in the same places.</p> <p>Example 2:</p>		

	<p>Input: s = "IYmpH"Output: "IYmpH"</p> <p>Explanation: There are no vowels in s (all characters in s are consonants), so we return "IYmpH".</p> <p>Constraints:</p> <p>1 <= s.length <= 10⁵</p> <p>s consists only of letters of the English alphabet in uppercase and lowercase.</p>		
2	Algorithm design		1
2.1	<p>Two pointers:</p> <p>Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.</p> <p>Example 1:</p> <p>Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]</p> <p>Output: 6</p>  <p>Explanation: The above elevation map (black section) is represented by an array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.</p> <p>Example 2:</p> <p>Input: height = [4,2,0,3,2,5] Output: 9</p> <p>Constraints:</p> <p>n == height.length</p> <p>1 <= n <= 2 * 10⁴</p> <p>0 <= height[i] <= 10⁵</p>	2	
2.2	<p>Next greater/smaller element:</p> <p>Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0 instead.</p> <p>Example 1:</p> <p>Input: temperatures = [73,74,75,71,69,72,76,73]Output: [1,1,4,2,1,1,0,0]</p> <p>Example 2:</p> <p>Input: temperatures = [30,40,50,60]Output: [1,1,1,0]</p>	2	

	<p>Example 3: Input: temperatures = [30,60,90]Output: [1,1,0]</p>		
2.3	<p>Sliding window: You have a bomb to defuse, and your time is running out! Your informer will provide you with a circular array code of length of n and a key k.</p> <p>To decrypt the code, you must replace every number. All the numbers are replaced simultaneously.</p> <p>If $k > 0$, replace the ith number with the sum of the next k numbers. If $k < 0$, replace the ith number with the sum of the previous k numbers. If $k == 0$, replace the ith number with 0. As code is circular, the next element of code[n-1] is code[0], and the previous element of code[0] is code[n-1].</p> <p>Given the circular array code and an integer key k, return the decrypted code to defuse the bomb!</p> <p>Example 1: Input: code = [5,7,1,4], k = 3 Output: [12,10,16,13] Explanation: Each number is replaced by the sum of the next 3 numbers. The decrypted code is [7+1+4, 1+4+5, 4+5+7, 5+7+1]. Notice that the numbers wrap around.</p> <p>Example 2: Input: code = [1,2,3,4], k = 0 Output: [0,0,0,0] Explanation: When k is zero, the numbers are replaced by 0.</p> <p>Example 3: Input: code = [2,4,9,3], k = -2 Output: [12,5,6,13] Explanation: The decrypted code is [3+9, 2+3, 4+2, 9+4]. Notice that the numbers wrap around again. If k is negative, the sum is of the previous numbers.</p> <p>Constraints: $n ==$ code.length $1 \leq n \leq 100$ $1 \leq \text{code}[i] \leq 100$ $-(n - 1) \leq k \leq n - 1$ </p>	2	
3	String processing		1,4
3.1	<p>Wildcard matching: Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:</p>	2	

	<p>'?' Matches any single character. '*' Matches any sequence of characters (including the empty sequence). The matching should cover the entire input string (not partial).</p> <p>Example 1: Input: s = "aa", p = "a" Output: false Explanation: "a" does not match the entire string "aa".</p> <p>Example 2: Input: s = "aa", p = "*" Output: true Explanation: "*" matches any sequence.</p> <p>Example 3: Input: s = "cb", p = "?a" Output: false Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.</p> <p>Constraints: 0 <= s.length, p.length <= 2000 s contains only lowercase English letters. p contains only lowercase English letters, '?' or '*'.</p>		
3.2	<p>Find Longest Awesome Substring You are given a string s. An awesome substring is a non-empty substring of s such that we can make any number of swaps in order to make it a palindrome.</p> <p>Return the length of the maximum length awesome substring of s.</p> <p>Example 1: Input: s = "3242415" Output: 5 Explanation: "24241" is the longest awesome substring, we can form the palindrome "24142" with some swaps.</p> <p>Example 2: Input: s = "12345678" Output: 1</p> <p>Example 3: Input: s = "213123" Output: 6 Explanation: "213123" is the longest awesome substring, we can form the palindrome "231132" with some swaps.</p> <p>Constraints: 1 <= s.length <= 105 s consists only of digits.</p>	2	

4	Efficient Range Query with Multiple Constraints		
	<p>Problem Statement: You are given a static array A of N integers. You need to efficiently answer Q queries of two types:</p> <ol style="list-style-type: none"> 1. Sum Query: Compute the sum of elements from index L to R (inclusive). 2. Minimum Query: Find the minimum element from index L to R (inclusive). <p>Additionally, you are required to handle the following bonus constraints:</p> <ul style="list-style-type: none"> • A can have up to 10^5 elements. • Q can go up to 10^5. • Modify the array at specific indices as part of query type 3: Update Query: Set $A[i]=X$. <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (size of the array) 2. Array A (space-separated integers). 3. Q (number of queries). 4. Each of the next Q lines describes a query: <ul style="list-style-type: none"> ○ Type 1: "1 L R" for sum query ○ Type 2: "2 L R" for minimum query ○ Type 3: "3 i X" for update query <p>Output Format:</p> <p>For each query of Type 1 and Type 2, output the result in a new line.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, Q \leq 10^5$ • $1 \leq A[i] \leq 10^9$ • $1 \leq L, R, i \leq N$ • $1 \leq X \leq 10^9$ <p>Example Input:</p> <pre> 5 3 8 6 7 1 5 1 2 4 </pre>		

	<p>2 3 5</p> <p>3 3 10</p> <p>1 1 5</p> <p>2 1 3</p> <p>Example Output:</p> <p>21</p> <p>1</p> <p>29</p> <p>3</p>		
5	<p>Advanced Tree Queries</p> <p>You are given a tree with N nodes. The tree is rooted at node 1. The following operations must be supported efficiently:</p> <ol style="list-style-type: none"> Find Ancestor Query: Given a node X and a number K, find the K-th ancestor of X. If it doesn't exist, return -1. Subtree Sum Query: Each node <i>iii</i> has a value V[i]. For a given node X, calculate the sum of values of all nodes in the subtree rooted at X. Lowest Common Ancestor Query: Given two nodes U and V, find their lowest common ancestor (LCA). Update Node Value: Update the value of a node X to Y. <p>Input Format:</p> <ol style="list-style-type: none"> N (number of nodes). N-1 lines describing the tree edges (undirected, 1-based indexing). Array V of N integers (values of nodes). Q (number of queries). Each of the next Q lines describes a query: <ul style="list-style-type: none"> Type 1: "1 X K" for K-th ancestor of X. Type 2: "2 X" for subtree sum of node X. Type 3: "3 U V" for LCA of nodes U and V. Type 4: "4 X Y" to update the value of node X to Y. <p>Output Format:</p> <p>For each query of Type 1, Type 2, and Type 3, output the result on a new line.</p> <p>Constraints:</p> <ul style="list-style-type: none"> $2 \leq N \leq 10^5$ $1 \leq V[i], Y \leq 10^9$ $1 \leq Q \leq 10^5$ 		

	<ul style="list-style-type: none"> • $1 \leq X, U, V, K \leq N$ <p>Example Input:</p> <pre> 5 1 2 1 3 3 4 3 5 3 8 6 7 1 6 1 5 2 2 3 3 4 5 4 3 10 2 3 1 4 3 </pre> <p>Example Output:</p> <pre> 1 19 3 26 -1 </pre>		
6	Shortest Path in a Weighted Successor Graph		
	<p>You are given a weighted directed graph with N nodes and M edges. Each node has a "successor," meaning a direct edge from the node to one specific other node. Additionally, you need to find the shortest path between two given nodes using successor relationships as well as the normal edges.</p> <p>You need to efficiently handle the following types of queries:</p> <ol style="list-style-type: none"> 1. Shortest Path Query: Find the shortest path between node U and node V, considering all edges (both successor and normal edges). 2. Update Successor Query: Change the successor of a node U to a new node SSS. <hr/> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of nodes), M (number of edges). 2. M lines with three integers A, B, W representing an edge from A to B with weight W. 3. N integers, where the i-th integer is the successor of node i. 4. Q (number of queries). 		

5. Each query is in one of the following formats:
- **Type 1:** "1 U V" to find the shortest path from U to V.
 - **Type 2:** "2 U S" to update the successor of node U to S.

Output Format:

For each query of **Type 1**, output the shortest path distance. If no path exists, output -1.

Constraints:

- $1 \leq N, M \leq 10^5$
- $1 \leq W \leq 10^9$
- $1 \leq Q \leq 10^5$
- The graph may contain cycles.
- Successors and edges may not cover all nodes.

Example Input:

5 6

1 2 2

1 3 4

2 4 7

3 4 3

4 5 1

3 5 5

2 3 5 4 1

4

1 1 5

2 3 2

1 1 5

1 5 3

	<p>Example Output:</p> <p>9</p> <p>8</p> <p>-1</p>		
7	<p>Multi-Tasking on a Complex Graph</p> <p>You are given a directed graph with N nodes and M edges. The graph may have cycles, and it can represent tasks with dependencies (DAG or general graph). You are required to perform the following operations efficiently:</p> <ol style="list-style-type: none"> 1. Find Strongly Connected Components (SCC): Identify all SCCs in the graph. 2. Check Bipartiteness of SCCs: Check whether each SCC is bipartite or not. 3. Single Source Shortest Path (SSSP): Find the shortest path from a given node SSS to a node TTT, considering edge weights. 4. Eulerian Path Check: Determine if the graph contains an Eulerian Path. 5. Update Edge Weights: Update the weight of a specific edge (U,V). <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of nodes), M (number of edges). 2. MMM lines, each containing three integers U,V,W, representing a directed edge from U to V with weight W. 3. Q (number of queries). 4. Each query is in one of the following formats: <ul style="list-style-type: none"> ○ Type 1: "1" to output the number of SCCs and whether each is bipartite. ○ Type 2: "2 S T" to find the shortest path from S to T. ○ Type 3: "3" to check if the graph contains an Eulerian Path. ○ Type 4: "4 U V W" to update the weight of edge (U,V) to W. <p>Output Format:</p> <ul style="list-style-type: none"> • For Type 1 queries: Output the number of SCCs and for each SCC, "YES" if it is bipartite, otherwise "NO". • For Type 2 queries: Output the shortest path distance. If no path exists, output -1. • For Type 3 queries: Output "YES" if an Eulerian Path exists, otherwise "NO". 		

	<ul style="list-style-type: none"> • Type 4 queries do not require an output. <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 10^5$. • $1 \leq W \leq 10^9$ • $1 \leq Q \leq 10^5$ • Graph may have self-loops and multiple edges. <p>Example Input:</p> <p>6 8</p> <p>1 2 3</p> <p>2 3 5</p> <p>3 1 2</p> <p>3 4 4</p> <p>4 5 1</p> <p>5 6 7</p> <p>6 4 6</p> <p>2 6 8</p> <p>5</p> <p>1</p> <p>2 1 5</p> <p>3</p> <p>4 3 4 10</p> <p>2 1 6</p> <p>Example Output:</p> <p>2 YES NO</p> <p>8</p> <p>YES</p> <p>13</p>		
8	Maximum Profit Job Scheduling		

	<p>You are given N jobs, where each job has:</p> <ul style="list-style-type: none"> • A start time $S[i]$, • An end time $E[i]$, • A profit $P[i]$. <p>Your task is to find the maximum profit you can achieve by scheduling non-overlapping jobs. You can only work on one job at a time.</p> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of jobs). 2. NNN lines, each containing three integers $S[i]$, $E[i]$, $P[i]$, representing the start time, end time, and profit of a job. <p>Output Format:</p> <p>Output a single integer: the maximum profit you can achieve.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N \leq 10^5$ • $1 \leq S[i], E[i], P[i] \leq 10^9$ • Jobs may overlap. <p>Example Input:</p> <pre> 5 1 3 50 3 5 20 6 19 100 2 100 200 8 10 70 </pre> <p>Example Output:</p> <pre> 250 </pre> <p>Explanation:</p> <ul style="list-style-type: none"> • Schedule job 4 ($S=2, E=100, P=200$). • Alternatively, schedule job 1 ($P=50$), job 3 ($P=100$), and job 5 ($P=70$). 		
--	---	--	--

	<ul style="list-style-type: none"> Maximum profit = 200+50. 		
9	<p>Maximum Path Sum in a Weighted Grid</p> <p>You are given a grid with N rows and M columns. Each cell (i,j) contains a non-negative weight W[i][j]. Your task is to find the maximum path sum from the top-left corner (1,1) to the bottom-right corner (N,M) with the following constraints:</p> <ol style="list-style-type: none"> You can only move down, right, or diagonally right-down at each step. Each cell can only be visited once during the path. <p>Input Format:</p> <ol style="list-style-type: none"> Two integers N and M representing the number of rows and columns in the grid. N lines, each containing M integers W[i][j], representing the weight of the grid cells. <p>Output Format:</p> <p>Output a single integer: the maximum path sum from (1,1) to (N,M).</p> <p>Constraints:</p> <ul style="list-style-type: none"> $1 \leq N, M \leq 1000$. $0 \leq W[i][j] \leq 10^4$ <p>Example Input:</p> <pre>3 3 1 2 3 4 5 6 7 8 9</pre> <p>Example Output:</p> <pre>21</pre> <p>Explanation:</p> <ul style="list-style-type: none"> One possible path: (1,1)→(2,2)→(3,3) with sum 1+5+9=15 The optimal path: (1,1)→(1,2)→(1,3)→(2,3)→(3,3) with sum 1+2+3+6+9=21 		

10	Optimal Delivery Routes with Dynamic Programming and Square Root Decomposition		
	<p>You are given a delivery system with N locations connected by bidirectional roads. Each road has a delivery cost. Your task is to process Q queries efficiently, where each query asks for the minimum delivery cost between two locations A and B.</p> <p>You need to optimize the solution using Square Root Decomposition and Dynamic Programming.</p> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of locations), M (number of roads), Q (number of queries). 2. M lines, each containing three integers U,V,C representing a road between U and V with cost C. 3. Q lines, each containing two integers A,B representing a query asking for the minimum delivery cost between locations A and B. <p>Output Format:</p> <p>For each query, output a single integer: the minimum delivery cost. If no route exists, output -1.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 10^5$ • $1 \leq C \leq 10^4$ • $1 \leq Q \leq 10^5$ <p>Example Input:</p> <pre> 5 6 3 1 2 4 2 3 2 3 4 6 1 5 10 4 5 1 3 5 7 1 4 </pre>		


	2 5 3 1 Example Output: 12 8 -1 Explanation: <ul style="list-style-type: none"> • Query 1: Minimum cost from 1→4 is 1→2→3→4 with cost 4+2+6=12. • Query 2: Minimum cost from 2→5 is 2→3→5 with cost 2+7=8. • Query 3: No valid path from 3→1, so output -1. 		
--	--	--	--

**Faculty of Technology and Engineering Devang Patel Institute
of Advance Technology and Research
Department of Computer Engineering**

Practical List

Academic Year	:	2024-25	Semester	:	6 th
Course code	:	CE397	Course name	:	Competitive Programming 2

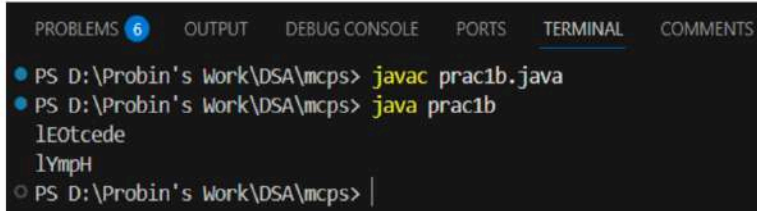
Sr. No.	Aim	Hrs	CO
1	Searching and sorting	2	4
1.1	<p>Ferris wheel: There are N children who want to go to a Ferris wheel in the form of an array arr[], and your task is to find a gondola for each child. Each gondola may have one or two children in it, and in addition, the total weight in a gondola may not exceed X. You know the weight of every child. What is the minimum number of gondolas needed for the children?</p> <p>Input: N = 4, X = 10, arr[] = {7, 2, 3, 9} Output: 3 Explanation: We need only 3 gondolas: {2, 3}, {7} and {9}.</p> <p>Input: N = 4, X = 6, arr[] = {2, 3, 3, 4} Output: 2 Explanation: We need only 2 gondolas: {2, 4} and {3, 3}</p> <p>Code:</p> <pre>import java.util.*; public class prac1a { public static int minGondolas(int N, int X, int[] arr) { Arrays.sort(arr); int left = 0, right = N - 1; int gondolas = 0; while (left <= right) { if (arr[left] + arr[right] <= X) { left++; } right--; gondolas++; } return gondolas; } }</pre>		

	<pre> } public static void main(String[] args) { int[] arr1 = {7, 2, 3, 9}; int X1 = 10; System.out.println(minGondolas(arr1.length, X1, arr1)); } } </pre> <p>Output:</p>  <pre> PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS PS D:\Probin's Work\DSA\mcps> javac prac1a.java PS D:\Probin's Work\DSA\mcps> java prac1a 3 PS D:\Probin's Work\DSA\mcps> </pre>		
1.2	<p>Sort Vowels in a String</p> <p>Given a 0-indexed string <i>s</i>, permute <i>s</i> to get a new string <i>t</i> such that:</p> <p>All consonants remain in their original places. More formally, if there is an index <i>i</i> with $0 \leq i < s.length$ such that <i>s</i>[<i>i</i>] is a consonant, then <i>t</i>[<i>i</i>] = <i>s</i>[<i>i</i>].</p> <p>The vowels must be sorted in the nondecreasing order of their ASCII values. More formally, for pairs of indices <i>i</i>, <i>j</i> with $0 \leq i < j < s.length$ such that <i>s</i>[<i>i</i>] and <i>s</i>[<i>j</i>] are vowels, then <i>t</i>[<i>i</i>] must not have a higher ASCII value than <i>t</i>[<i>j</i>].</p> <p>Return the resulting string.</p> <p>The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in lowercase or uppercase. Consonants comprise all letters that are not vowels.</p> <p>Example 1:</p> <p>Input: <i>s</i> = "lEetcOde"</p> <p>Output: "lEOtcede"</p> <p>Explanation: 'E', 'O', and 'e' are the vowels in <i>s</i>; 'l', 't', 'c', and 'd' are all consonants. The vowels are sorted according to their ASCII values, and the consonants remain in the same places.</p> <p>Example 2:</p>		

	<p>Input: s = "IYmpH"Output: "IYmpH"</p> <p>Explanation: There are no vowels in s (all characters in s are consonants), so we return "IYmpH".</p> <p>Constraints:</p> <p>1 <= s.length <= 10⁵ s consists only of letters of the English alphabet in uppercase and lowercase.</p> <p>Code:</p> <pre>import java.util.*; public class prac1b { public static String sortVowelsInString(String s) { List<Character> vowels = new ArrayList<>(); for (char c : s.toCharArray()) { if (isVowel(c)) { vowels.add(c); } } Collections.sort(vowels); StringBuilder result = new StringBuilder(); int vowelIndex = 0; for (char c : s.toCharArray()) { if (isVowel(c)) { result.append(vowels.get(vowelIndex++)); } else { result.append(c); } } return result.toString(); } private static boolean isVowel(char c) { c = Character.toLowerCase(c); return c == 'a' c == 'e' c == 'i' c == 'o' c == 'u'; } public static void main(String[] args) { System.out.println(sortVowelsInString("IEetcOde")); System.out.println(sortVowelsInString("IYmpH")); } }</pre>		
--	---	--	--


```
}  
}
```

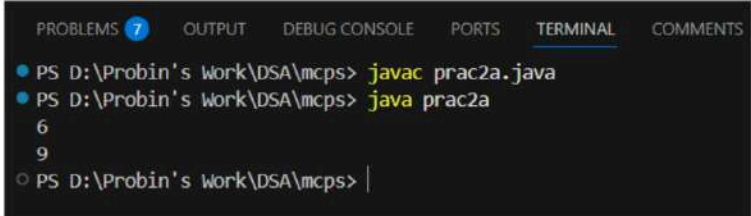
Output:



The screenshot shows an IDE terminal window with the following tabs: PROBLEMS (6), OUTPUT, DEBUG CONSOLE, PORTS, TERMINAL, and COMMENTS. The terminal output is as follows:

```
PS D:\Probin's Work\DSA\mcps> javac prac1b.java  
PS D:\Probin's Work\DSA\mcps> java prac1b  
lE0tcede  
lYmpH  
PS D:\Probin's Work\DSA\mcps> |
```

2	Algorithm design		1
2.1	<p>Two pointers:</p> <p>Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.</p> <p>Example 1: Input: height = [0,1,0,2,1,0,1,3,2,1,2,1] Output: 6</p>  <p>Explanation: The above elevation map (black section) is represented by an array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.</p> <p>Example 2: Input: height = [4,2,0,3,2,5] Output: 9</p> <p>Constraints:</p> <p>n == height.length 1 <= n <= 2 * 10⁴ 0 <= height[i] <= 10⁵</p> <p>Code:</p> <pre>import java.util.*; public class prac2a { public int trap(int[] height) { if (height == null height.length <= 2) { return 0; } int left = 0, right = height.length - 1; int left_max = 0, right_max = 0; int waterTrapped = 0; while (left <= right) { if (height[left] <= height[right]) { if (height[left] >= left_max) { left_max = height[left]; } else { waterTrapped += left_max - height[left]; } left++; } else { if (height[right] >= right_max) { right_max = height[right]; } else { waterTrapped += right_max - height[right]; } right--; } } return waterTrapped; } }</pre>	2	

	<pre> if (height[right] >= right_max) { right_max = height[right]; } else { waterTrapped += right_max - height[right]; } right--; } } return waterTrapped; } public static void main(String[] args) { prac2a solution = new prac2a(); int[] height1 = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1}; System.out.println(solution.trap(height1)); int[] height2 = {4, 2, 0, 3, 2, 5}; System.out.println(solution.trap(height2)); } } </pre> <p>Output:</p>  <pre> PROBLEMS 7 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS PS D:\Probin's Work\DSA\mcps> javac prac2a.java PS D:\Probin's Work\DSA\mcps> java prac2a 6 9 PS D:\Probin's Work\DSA\mcps> </pre>		
2.2	<p>Next greater/smaller element:</p> <p>Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is thenumber of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this ispossible, keep answer[i] == 0 instead.</p> <p>Example 1: Input: temperatures = [73,74,75,71,69,72,76,73]Output: [1,1,4,2,1,1,0,0]</p> <p>Example 2: Input: temperatures = [30,40,50,60]Output: [1,1,1,0]</p>	2	
	<p>Example 3: Input: temperatures = [30,60,90]Output: [1,1,0]</p>		

Code:

```

import java.util.*;

public class prac2b {
    public int[] dailyTemperatures(int[] temperatures) {
        int n = temperatures.length;
        int[] answer = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && temperatures[i] >
temperatures[stack.peek()]) {
                int idx = stack.pop();
                answer[idx] = i - idx;
            }
            stack.push(i);
        }

        return answer;
    }

    public static void main(String[] args) {
        prac2b sol = new prac2b();

        int[] temperatures1 = {73, 74, 75, 71, 69, 72, 76, 73};
        System.out.println(Arrays.toString(sol.dailyTemperatures(te
mperatures1)));

        int[] temperatures2 = {30, 40, 50, 60};
        System.out.println(Arrays.toString(sol.dailyTemperatures(te
mperatures2)));

        int[] temperatures3 = {30, 60, 90};
        System.out.println(Arrays.toString(sol.dailyTemperatures(te
mperatures3)));
    }
}

```

Output:

```

PROBLEMS 6 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS
● PS D:\Probin's Work\DSA\mcps> javac prac2b.java
● PS D:\Probin's Work\DSA\mcps> java prac2b
[1, 1, 4, 2, 1, 1, 0, 0]
[1, 1, 1, 0]
[1, 1, 0]
○ PS D:\Probin's Work\DSA\mcps> |

```


2.3	<p>Sliding window: You have a bomb to defuse, and your time is running out! Your informer will provide you with a circular array code of length of n and a key k.</p> <p>To decrypt the code, you must replace every number. All the numbers are replaced simultaneously.</p> <p>If $k > 0$, replace the ith number with the sum of the next k numbers. If $k < 0$, replace the ith number with the sum of the previous k numbers. If $k == 0$, replace the ith number with 0.</p> <p>As code is circular, the next element of code[n-1] is code[0], and the previous element of code[0] is code[n-1].</p> <p>Given the circular array code and an integer key k, return the decrypted code to defuse the bomb!</p> <p>Example 1: Input: code = [5,7,1,4], k = 3 Output: [12,10,16,13] Explanation: Each number is replaced by the sum of the next 3 numbers. The decrypted code is [7+1+4, 1+4+5, 4+5+7, 5+7+1]. Notice that the numbers wrap around.</p> <p>Example 2: Input: code = [1,2,3,4], k = 0 Output: [0,0,0,0] Explanation: When k is zero, the numbers are replaced by 0.</p> <p>Example 3: Input: code = [2,4,9,3], k = -2 Output: [12,5,6,13] Explanation: The decrypted code is [3+9, 2+3, 4+2, 9+4]. Notice that the numbers wrap around again. If k is negative, the sum is of the previous numbers.</p> <p>Constraints: $n ==$ $code.length$ $1 \leq n \leq 100$ $1 \leq code[i] \leq 100$ $-(n - 1) \leq k \leq n - 1$</p> <p>Code:</p> <pre>import java.util.*; import java.util.*; public class prac2c { public int[] decrypt(int[] code, int k) { int n = code.length;</pre>	2
-----	--	---

```

int[] result = new int[n];

if (k == 0) {
    return result;
}

for (int i = 0; i < n; i++) {
    int sum = 0;
    if (k > 0) {
        for (int j = 1; j <= k; j++) {
            sum += code[(i + j) % n];
        }
    } else {
        for (int j = 1; j <= -k; j++) {
            sum += code[(i - j + n) % n];
        }
    }
    result[i] = sum;
}

return result;
}

public static void main(String[] args) {
    prac2c solution = new prac2c();

    int[] code1 = {5, 7, 1, 4};
    int k1 = 3;
    System.out.println(Arrays.toString(solution.decrypt(code1,
k1)));

    int[] code2 = {1, 2, 3, 4};
    int k2 = 0;
    System.out.println(Arrays.toString(solution.decrypt(code2,
k2)));

    int[] code3 = {2, 4, 9, 3};
    int k3 = -2;
    System.out.println(Arrays.toString(solution.decrypt(code3,
k3)));
}
}

```

Output:

```

PROBLEMS 7 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS
● PS D:\Probin's Work\DSA\mcps> javac prac2c.java
● PS D:\Probin's Work\DSA\mcps> java prac2c
[12, 10, 16, 13]
[0, 0, 0, 0]
[12, 5, 6, 13]
○ PS D:\Probin's Work\DSA\mcps> |

```

3	String processing		1,4
3.1	Wildcard matching: Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:	2	

'?' Matches any single character.
 '*' Matches any sequence of characters (including the empty sequence).
 The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p =

"a" Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p =

"*" Output: true

Explanation: '*' matches any sequence.

Example 3:

Input: s = "cb", p =

"?a" Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

Constraints:

0 <= s.length, p.length <= 2000

s contains only lowercase English letters.

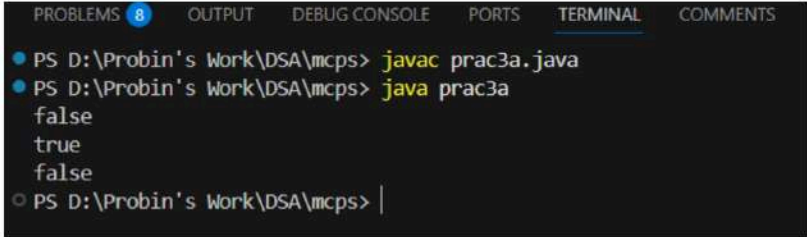
p contains only lowercase English letters, '?' or '*'.

Code:

```
import java.util.*;
public class prac3a {
    public boolean isMatch(String s, String p) {
        int m = s.length(), n = p.length();
        boolean[][] dp = new boolean[m + 1][n + 1];
        dp[0][0] = true;

        for (int j = 1; j <= n; j++) {
            if (p.charAt(j - 1) == '*') {
                dp[0][j] = dp[0][j - 1];
            }
        }

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (p.charAt(j - 1) == s.charAt(i - 1) || p.charAt(j - 1) == '?') {
                    dp[i][j] = dp[i - 1][j - 1];
                } else if (p.charAt(j - 1) == '*') {
                    dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
                }
            }
        }
    }
}
```

	<pre> } return dp[m][n]; } public static void main(String[] args) { prac3a solution = new prac3a(); System.out.println(solution.isMatch("aa", "a")); System.out.println(solution.isMatch("aa", "*")); System.out.println(solution.isMatch("cb", "?a")); } } </pre> <p>Output:</p> 		
3.2	<p>Find Longest Awesome Substring</p> <p>You are given a string s. An awesome substring is a non-empty substring of s such that we can make any number of swaps in order to make it a palindrome.</p> <p>Return the length of the maximum length awesome substring of s.</p> <p>Example 1: Input: $s = "3242415"$ Output: 5 Explanation: "24241" is the longest awesome substring, we can form the palindrome "24142" with some swaps.</p> <p>Example 2: Input: $s = "12345678"$ Output: 1</p> <p>Example 3: Input: $s = "213123"$ Output: 6 Explanation: "213123" is the longest awesome substring, we can form the palindrome "231132" with some swaps.</p> <p>Constraints: $1 \leq s.length \leq 105$ s consists only of digits.</p>	2	

Code:

```

import java.util.*;
public class prac3b {
    public int longestAwesome(String s) {
        int n = s.length();
        int maxLength = 1;
        int[] seen = new int[1024];
        seen[0] = -1;
        int mask = 0;

        for (int i = 0; i < n; i++) {
            mask ^= 1 << (s.charAt(i) - '0');
            if (seen[mask] != 0) {
                maxLength = Math.max(maxLength, i - seen[mask]);
            }
            for (int j = 0; j < 10; j++) {
                int tempMask = mask ^ (1 << j);
                if (seen[tempMask] != 0) {
                    maxLength = Math.max(maxLength, i -
seen[tempMask]);
                }
            }
            seen[mask] = i + 1;
        }

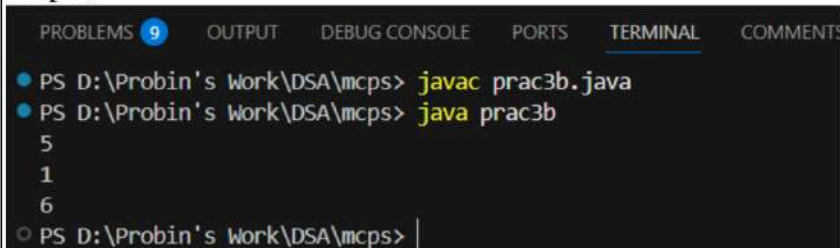
        return maxLength;
    }

    public static void main(String[] args) {
        prac3b solution = new prac3b();

        System.out.println(solution.longestAwesome("3242415"));
        System.out.println(solution.longestAwesome("12345678"));
        System.out.println(solution.longestAwesome("213123"));
    }
}

```

Output;



```

PROBLEMS 9 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS
• PS D:\Probin's Work\DSA\mcps> javac prac3b.java
• PS D:\Probin's Work\DSA\mcps> java prac3b
5
1
6
○ PS D:\Probin's Work\DSA\mcps> |

```

4	Efficient Range Query with Multiple Constraints		
	<p>Problem Statement: You are given a static array A of N integers. You need to efficiently answer Q queries of two types:</p> <ol style="list-style-type: none"> 1. Sum Query: Compute the sum of elements from index L to R (inclusive). 2. Minimum Query: Find the minimum element from index L to R (inclusive). <p>Additionally, you are required to handle the following bonus constraints:</p> <ul style="list-style-type: none"> • A can have up to 10^5 elements. • Q can go up to 10^5. • Modify the array at specific indices as part of query type 3: Update Query: Set $A[i]=X$. <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (size of the array) 2. Array A (space-separated integers). 3. Q (number of queries). 4. Each of the next Q lines describes a query: <ul style="list-style-type: none"> ○ Type 1: "1 L R" for sum query ○ Type 2: "2 L R" for minimum query ○ Type 3: "3 i X" for update query <p>Output Format:</p> <p>For each query of Type 1 and Type 2, output the result in a new line.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, Q \leq 10^5$ • $1 \leq A[i] \leq 10^9$ • $1 \leq L, R, i \leq N$ • $1 \leq X \leq 10^9$ <p>Example</p> <p>Input:</p> <pre> 5 3 8 6 7 1 5 </pre>		

	1 2 4		
--	-------	--	--

2 3 5

3 3 10

1 1 5

2 1 3

Example Output:

21

1

29

3

Code:

```
import java.util.*;

public class prac4 {

    public static int sumQuery(int arr[], int start, int end) {
        int sum = 0;
        for (int i = start; i <= end; i++) {
            sum += arr[i];
        }
        return sum;
    }

    public static int minQuery(int arr[], int start, int end) {
        int min = arr[start];
        for (int i = start + 1; i <= end; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        return min;
    }

    public static void updateQuery(int arr[], int index, int value) {
        arr[index] = value;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();
        int[] arr = new int[n];
```

```

    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();
    }

    int q = sc.nextInt();

    for (int i = 0; i < q; i++) {
        int type = sc.nextInt();

        if (type == 1) {
            int L = sc.nextInt();
            int R = sc.nextInt();
            System.out.println(sumQuery(arr, L - 1, R - 1));
        } else if (type == 2) {
            int L = sc.nextInt();
            int R = sc.nextInt();
            System.out.println(minQuery(arr, L - 1, R - 1));
        } else if (type == 3) {
            int idx = sc.nextInt();
            int value = sc.nextInt();
            updateQuery(arr, idx - 1, value);
        }
    }
}

```

Output:

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS D:\Probin's Work\DSA\mcps> javac prac4.java
PS D:\Probin's Work\DSA\mcps> java prac4
5
3 8 6 7 1
5
1 2 4
21
2 3 5
1
3 3 10
1 1 5
29
2 1 3
3
PS D:\Probin's Work\DSA\mcps>

```

5**Advanced Tree Queries**

	<p>You are given a tree with N nodes. The tree is rooted at node 1. The following operations must be supported efficiently:</p> <ol style="list-style-type: none"> 1. Find Ancestor Query: Given a node X and a number K, find the K-th ancestor of X. If it doesn't exist, return -1. 2. Subtree Sum Query: Each node i has a value $V[i]$. For a given node X, calculate the sum of values of all nodes in the subtree rooted at X. 3. Lowest Common Ancestor Query: Given two nodes U and V, find their lowest common ancestor (LCA). 4. Update Node Value: Update the value of a node X to Y. <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of nodes). 2. $N-1$ lines describing the tree edges (undirected, 1-based indexing). 3. Array V of N integers (values of nodes). 4. Q (number of queries). 5. Each of the next Q lines describes a query: <ul style="list-style-type: none"> ○ Type 1: "1 X K" for K-th ancestor of X. ○ Type 2: "2 X" for subtree sum of node X. ○ Type 3: "3 U V" for LCA of nodes U and V. ○ Type 4: "4 X Y" to update the value of node X to Y. <p>Output Format:</p> <p>For each query of Type 1, Type 2, and Type 3, output the result on a new line.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $2 \leq N \leq 10^5$ • $1 \leq V[i], Y \leq 10^9$ • $1 \leq Q \leq 10^5$ 		
--	--	--	--

- $1 \leq X, U, V, K \leq N$

Example Input:

```
5
1 2
1 3
3 4
3 5
3 8 6 7 1
6
1 5 2
2 3
3 4 5
4 3 10
2 3
1 4 3
```

Example Output:

```
1
19
3
26
-1
```

Code:

```
import java.util.*;
import java.util.*;

public class prac5 {

    static class Tree {
        int[] parent, depth, subtreeSum, value;
        int[][] up;
        List<List<Integer>> adj;

        public Tree(int n) {
            parent = new int[n + 1];
            depth = new int[n + 1];
            value = new int[n + 1];
            subtreeSum = new int[n + 1];
            adj = new ArrayList<>();
            up = new int[n + 1][21];
            for (int i = 0; i <= n; i++) adj.add(new ArrayList<>());
        }

        public void dfs(int u, int p, int d) {
            parent[u] = p;
```

```

    depth[u] = d;
    subtreeSum[u] = value[u];
    for (int i : adj.get(u)) {
        if (i != p) {
            dfs(i, u, d + 1);
            subtreeSum[u] += subtreeSum[i];
        }
    }
}

public void preprocess(int n) {
    dfs(1, -1, 0);
    for (int i = 1; i <= n; i++) up[i][0] = parent[i];
    for (int j = 1; j <= 20; j++) {
        for (int i = 1; i <= n; i++) {
            if (up[i][j - 1] != -1) {
                up[i][j] = up[up[i][j - 1]][j - 1];
            }
        }
    }
}

public int kthAncestor(int x, int k) {
    for (int i = 20; i >= 0; i--) {
        if ((k & (1 << i)) != 0) {
            x = up[x][i];
            if (x == -1) return -1;
        }
    }
    return x;
}

public int lca(int u, int v) {
    if (depth[u] < depth[v]) {
        int temp = u;
        u = v;
        v = temp;
    }
    for (int i = 20; i >= 0; i--) {
        if (depth[u] - (1 << i) >= depth[v]) {
            u = up[u][i];
        }
    }
    if (u == v) return u;
    for (int i = 20; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
}

```

```

    }
    return parent[u];
}

public void updateValue(int u, int y) {
    value[u] = y;
    dfs(1, -1, 0);
}

public int subtreeSum(int u) {
    return subtreeSum[u];
}
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    Tree tree = new Tree(n);

    for (int i = 1; i < n; i++) {
        int u = sc.nextInt();
        int v = sc.nextInt();
        tree.adj.get(u).add(v);
        tree.adj.get(v).add(u);
    }

    for (int i = 1; i <= n; i++) {
        tree.value[i] = sc.nextInt();
    }

    tree.preprocess(n);

    int q = sc.nextInt();
    for (int i = 0; i < q; i++) {
        int type = sc.nextInt();
        if (type == 1) {
            int x = sc.nextInt();
            int k = sc.nextInt();
            System.out.println(tree.kthAncestor(x, k));
        } else if (type == 2) {
            int x = sc.nextInt();
            System.out.println(tree.subtreeSum(x));
        } else if (type == 3) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            System.out.println(tree.lca(u, v));
        } else if (type == 4) {
            int x = sc.nextInt();
            int y = sc.nextInt();

```



```

        tree.updateValue(x, y);
    }
}
}
}

```

Output:

```

PS D:\Probin's Work\DSA\mcps> javac prac5.java
PS D:\Probin's Work\DSA\mcps> java prac5
5
1 2
1 3
3 4
3 5
3 8 6 7 1
6
1 5 2
1
2 3
14
3 4 5
3
4 3 10
2 3
18
1 4 3
-1
PS D:\Probin's Work\DSA\mcps> |

```

	<p>You are given a weighted directed graph with N nodes and M edges. Each node has a "successor," meaning a direct edge from the node to one specific other node. Additionally, you need to find the shortest path between two given nodes using successor relationships as well as the normal edges.</p> <p>You need to efficiently handle the following types of queries:</p> <ol style="list-style-type: none"> 1. Shortest Path Query: Find the shortest path between node U and node V, considering all edges (both successor and normal edges). 2. Update Successor Query: Change the successor of a node U to a new node SSS. <hr/> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of nodes), M (number of edges). 2. M lines with three integers A,B,W representing an edge from A to B with weight W. 3. N integers, where the i-th integer is the successor of node i. 4. Q (number of queries). 		
--	---	--	--

	<p>5. Each query is in one of the following formats:</p> <ul style="list-style-type: none"> ○ Type 1: "1 U V" to find the shortest path from U to V. ○ Type 2: "2 U S" to update the successor of node U to S. <hr/> <p>Output Format:</p> <p>For each query of Type 1, output the shortest path distance. If no path exists, output -1.</p> <hr/> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 10^5$ • $1 \leq W \leq 10^9$ • $1 \leq Q \leq 10^5$ • The graph may contain cycles. • Successors and edges may not cover all nodes. <p>Example Input:</p> <pre> 5 6 1 2 2 1 3 4 2 4 7 3 4 3 4 5 1 3 5 5 2 3 5 4 1 4 1 1 5 2 3 2 1 1 5 1 5 3 </pre>		
--	--	--	--

Example Output:

9

8

-1

Code:

```

import java.util.*;

public class prac6 {
    static class Edge {
        int to, weight;
        Edge(int to, int weight) {
            this.to = to;
            this.weight = weight;
        }
    }

    static long dijkstra(int start, int end, List<List<Edge>> graph,
int[] successors, int n) {
        long[] dist = new long[n + 1];
        Arrays.fill(dist, Long.MAX_VALUE);
        PriorityQueue<long[]> pq = new
PriorityQueue<>(Comparator.comparingLong(a -> a[1]));

        dist[start] = 0;
        pq.offer(new long[]{start, 0});

        while (!pq.isEmpty()) {
            long[] curr = pq.poll();
            int node = (int)curr[0];
            long d = curr[1];

            if (d > dist[node]) continue;
            if (node == end) return d;

            for (Edge e : graph.get(node)) {
                long newDist = dist[node] + e.weight;
                if (newDist < dist[e.to]) {
                    dist[e.to] = newDist;
                    pq.offer(new long[]{e.to, newDist});
                }
            }

            int succ = successors[node];
            if (succ != 0) {
                long newDist = dist[node];
                if (newDist < dist[succ]) {

```

```

        dist[succ] = newDist;
        pq.offer(new long[]{succ, newDist});
    }
}
return dist[end] == Long.MAX_VALUE ? -1 : dist[end];
}

public static void main(String[] args) {

    int n = 5;
    int m = 6;

    List<List<Edge>> graph = new ArrayList<>();
    for (int i = 0; i <= n; i++) {
        graph.add(new ArrayList<>());
    }

    int[][] edges = {
        {1, 2, 2},
        {1, 3, 4},
        {2, 4, 7},
        {3, 4, 3},
        {4, 5, 1},
        {3, 5, 5}
    };

    for (int[] edge : edges) {
        graph.get(edge[0]).add(new Edge(edge[1], edge[2]));
    }


    int[] successors = new int[n + 1];
    int[] succInput = {0, 2, 3, 5, 4, 1};
    for (int i = 1; i <= n; i++) {
        successors[i] = succInput[i];
    }

    int[][] queries = {
        {1, 1, 5},
        {2, 3, 2},
        {1, 1, 5},
        {1, 5, 3}
    };

    for (int[] query : queries) {
        int type = query[0];
        int u = query[1];

        if (type == 1) {

```

	<pre> int v = query[2]; long result = dijkstra(u, v, graph, successors, n); System.out.println(result); } else { int s = query[2]; successors[u] = s; } } } } </pre> <p>Output:</p>  <pre> PS D:\Probin's Work\DSA\mcps> javac prac6.java PS D:\Probin's Work\DSA\mcps> java prac6 0 4 0 PS D:\Probin's Work\DSA\mcps> </pre>		
7	Multi-Tasking on a Complex Graph		

	<p>You are given a directed graph with N nodes and M edges. The graph may have cycles, and it can represent tasks with dependencies (DAG or general graph). You are required to perform the following operations efficiently:</p> <ol style="list-style-type: none"> 1. Find Strongly Connected Components (SCC): Identify all SCCs in the graph. 2. Check Bipartiteness of SCCs: Check whether each SCC is bipartite or not. 3. Single Source Shortest Path (SSSP): Find the shortest path from a given node SSS to a node TTT, considering edge weights. 4. Eulerian Path Check: Determine if the graph contains an Eulerian Path. 5. Update Edge Weights: Update the weight of a specific edge (U,V). <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of nodes), M (number of edges). 2. MMM lines, each containing three integers U,V,W, representing a directed edge from U to V with weight W. 3. Q (number of queries). 4. Each query is in one of the following formats: <ul style="list-style-type: none"> ○ Type 1: "1" to output the number of SCCs and whether each is bipartite. ○ Type 2: "2 S T" to find the shortest path from S to T. ○ Type 3: "3" to check if the graph contains an Eulerian Path. ○ Type 4: "4 U V W" to update the weight of edge (U,V) to W. <p>Output Format:</p> <ul style="list-style-type: none"> • For Type 1 queries: Output the number of SCCs and for each SCC, "YES" if it is bipartite, otherwise "NO". • For Type 2 queries: Output the shortest path distance. If no path exists, output -1. • For Type 3 queries: Output "YES" if an Eulerian Path exists, otherwise "NO". 		
--	---	--	--

	<ul style="list-style-type: none"> • Type 4 queries do not require an output. <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 10^5$. • $1 \leq W \leq 10^9$ • $1 \leq Q \leq 10^5$ • Graph may have self-loops and multiple edges. <p>Example Input: 6</p> <p>8</p> <p>1 2 3</p> <p>2 3 5</p> <p>3 1 2</p> <p>3 4 4</p> <p>4 5 1</p> <p>5 6 7</p> <p>6 4 6</p> <p>2 6 8</p> <p>5</p> <p>1</p> <p>2 1 5</p> <p>3</p> <p>4 3 4 10</p> <p>2 1 6</p> <p>Example Output:</p> <p>2 YES NO</p> <p>8</p> <p>YES 13</p>		
--	---	--	--

Code:

```

import java.util.*;

class Graph {
    static final int INF = Integer.MAX_VALUE;
    int N, M;
    List<Edge>[] adjList;
    List<Edge>[] revAdjList;
    int[] dist;
    int[] color;
    int[] inDegree;
    int[] outDegree;
    int[] parent;

    static class Edge {
        int dest, weight;
        Edge(int dest, int weight) {
            this.dest = dest;
            this.weight = weight;
        }
    }

    Graph(int N, int M) {
        this.N = N;
        this.M = M;
        adjList = new ArrayList[N + 1];
        revAdjList = new ArrayList[N + 1];
        for (int i = 0; i <= N; i++) {
            adjList[i] = new ArrayList<>();
            revAdjList[i] = new ArrayList<>();
        }
        dist = new int[N + 1];
        color = new int[N + 1];
        inDegree = new int[N + 1];
        outDegree = new int[N + 1];
        parent = new int[N + 1];
    }

    void addEdge(int u, int v, int w) {
        adjList[u].add(new Edge(v, w));
        revAdjList[v].add(new Edge(u, w));
    }

    void updateEdge(int u, int v, int w) {
        for (Edge e : adjList[u]) {
            if (e.dest == v) {

```

```

        e.weight = w;
        break;
    }
}

void dfs(int node, boolean[] visited, Stack<Integer> stack) {
    visited[node] = true;
    for (Edge e : adjList[node]) {
        if (!visited[e.dest]) {
            dfs(e.dest, visited, stack);
        }
    }
    stack.push(node);
}

void revDfs(int node, boolean[] visited, List<Integer> scc) {
    visited[node] = true;
    scc.add(node);
    for (Edge e : revAdjList[node]) {
        if (!visited[e.dest]) {
            revDfs(e.dest, visited, scc);
        }
    }
}

void findSCCs() {
    boolean[] visited = new boolean[N + 1];
    Stack<Integer> stack = new Stack<>();
    for (int i = 1; i <= N; i++) {
        if (!visited[i]) {
            dfs(i, visited, stack);
        }
    }

    Arrays.fill(visited, false);
    List<List<Integer>> sccs = new ArrayList<>();
    while (!stack.isEmpty()) {
        int node = stack.pop();
        if (!visited[node]) {
            List<Integer> scc = new ArrayList<>();
            revDfs(node, visited, scc);
            sccs.add(scc);
        }
    }

    System.out.println(sccs.size());
    for (List<Integer> scc : sccs) {
        boolean isBipartite = checkBipartite(scc);
    }
}

```

```

        System.out.println(isBipartite ? "YES" : "NO");
    }
}

boolean checkBipartite(List<Integer> scc) {
    Arrays.fill(color, -1);
    for (int node : scc) {
        if (color[node] == -1 && !bfsBipartite(node)) {
            return false;
        }
    }
    return true;
}

boolean bfsBipartite(int start) {
    Queue<Integer> queue = new LinkedList<>();
    color[start] = 0;
    queue.add(start);
    while (!queue.isEmpty()) {
        int node = queue.poll();
        for (Edge e : adjList[node]) {
            int nextNode = e.dest;
            if (color[nextNode] == -1) {
                color[nextNode] = 1 - color[node];
                queue.add(nextNode);
            } else if (color[nextNode] == color[node]) {
                return false;
            }
        }
    }
    return true;
}

void dijkstra(int start) {
    Arrays.fill(dist, INF);
    dist[start] = 0;
    PriorityQueue<int[]> pq = new
PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
    pq.add(new int[]{start, 0});
    while (!pq.isEmpty()) {
        int[] node = pq.poll();
        int u = node[0], d = node[1];
        if (d > dist[u]) continue;
        for (Edge e : adjList[u]) {
            int v = e.dest, w = e.weight;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.add(new int[]{v, dist[v]});
            }
        }
    }
}

```

```

    }
}

boolean isEulerianPath() {
    int startCount = 0, endCount = 0;
    for (int i = 1; i <= N; i++) {
        if (inDegree[i] == outDegree[i] + 1) {
            startCount++;
        } else if (outDegree[i] == inDegree[i] + 1) {
            endCount++;
        } else if (inDegree[i] != outDegree[i]) {
            return false;
        }
    }
    return (startCount == 1 && endCount == 1) || (startCount == 0
&& endCount == 0);
}

public class prac7 {
    public static void main(String[] args) {
        Graph graph = new Graph(6, 8);
        graph.addEdge(1, 2, 3);
        graph.addEdge(2, 3, 5);
        graph.addEdge(3, 1, 2);
        graph.addEdge(3, 4, 4);
        graph.addEdge(4, 5, 1);
        graph.addEdge(5, 6, 7);
        graph.addEdge(6, 4, 6);
        graph.addEdge(2, 6, 8);

        int Q = 5;

        graph.findSCCs();

        graph.dijkstra(1);
        System.out.println(graph.dist[5] == Graph.INF ? -1 :
graph.dist[5]);

        System.out.println(graph.isEulerianPath() ? "YES" : "NO");

        graph.updateEdge(3, 4, 10);

        graph.dijkstra(1);
        System.out.println(graph.dist[6] == Graph.INF ? -1 :
graph.dist[6]);
    }
}

```

Output:

```
PROBLEMS 16 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS

PS D:\Probin's Work\DSA\mcps> javac -Xlint:unchecked prac7.java
prac7.java:25: warning: [unchecked] unchecked conversion
    adjList = new ArrayList[N + 1];
    ^
    required: List<Edge>[]
    found:    ArrayList[]
prac7.java:26: warning: [unchecked] unchecked conversion
    revAdjList = new ArrayList[N + 1];
    ^
    required: List<Edge>[]
    found:    ArrayList[]
2 warnings
● PS D:\Probin's Work\DSA\mcps> java prac7
2
NO
NO
13
YES
11
○ PS D:\Probin's Work\DSA\mcps> |
```

8	Maximum Profit Job Scheduling		
	<p>You are given N jobs, where each job has:</p> <ul style="list-style-type: none"> • A start time $S[i]$, • An end time $E[i]$, • A profit $P[i]$. <p>Your task is to find the maximum profit you can achieve by scheduling non-overlapping jobs. You can only work on one job at a time.</p> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of jobs). 2. NNN lines, each containing three integers $S[i]$, $E[i]$, $P[i]$, representing the start time, end time, and profit of a job. <p>Output Format:</p> <p>Output a single integer: the maximum profit you can achieve.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N \leq 10^5$ • $1 \leq S[i], E[i], P[i] \leq 10^9$ • Jobs may overlap. <p>Example Input:</p> <pre> 5 1 3 50 3 5 20 6 19 100 2 100 200 8 10 70 </pre> <p>Example Output:</p> <pre> 250 </pre> <p>Explanation:</p> <ul style="list-style-type: none"> • Schedule job 4 ($S=2, E=100, P=200$). • Alternatively, schedule job 1 ($P=50$), job 3 ($P=100$), and job 5 ($P=70$). 		

- Maximum profit = 200+50.

Code:

```
import java.io.*;
import java.util.*;

public class prac8 {
    static class Job {
        int start, end, profit;

        Job(int start, int end, int profit) {
            this.start = start;
            this.end = end;
            this.profit = profit;
        }
    }

    public static void main(String[] args) {
        int n = 5;
        Job[] jobs = new Job[n];

        jobs[0] = new Job(1, 3, 50);
        jobs[1] = new Job(3, 5, 20);
        jobs[2] = new Job(6, 19, 100);
        jobs[3] = new Job(2, 100, 200);
        jobs[4] = new Job(8, 10, 70);

        Arrays.sort(jobs, (a, b) -> a.end - b.end);

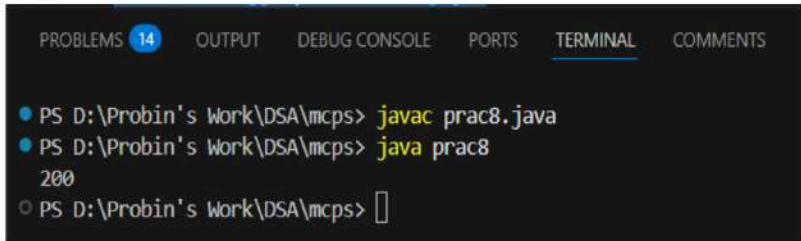
        int[] dp = new int[n];
        dp[0] = jobs[0].profit;

        for (int i = 1; i < n; i++) {
            int l = 0, r = i - 1;
            int max = 0;

            while (l <= r) {
                int mid = (l + r) / 2;

                if (jobs[mid].end <= jobs[i].start) {
                    max = Math.max(max, dp[mid]);
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }

            dp[i] = Math.max(jobs[i].profit, max + jobs[i].profit);
        }
    }
}
```


	<pre> } int maxProfit = 0; for (int i = 0; i < n; i++) { maxProfit = Math.max(maxProfit, dp[i]); } System.out.println(maxProfit); } } </pre> <p>Output:</p>  <pre> PROBLEMS 14 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS • PS D:\Probin's Work\DSA\mcps> javac prac8.java • PS D:\Probin's Work\DSA\mcps> java prac8 200 ○ PS D:\Probin's Work\DSA\mcps> </pre>		
9	Maximum Path Sum in a Weighted Grid		

	<p>You are given a grid with N rows and M columns. Each cell (i,j) contains a non-negative weight $W[i][j]$. Your task is to find the maximum path sum from the top-left corner (1,1) to the bottom-right corner (N,M) with the following constraints:</p> <ol style="list-style-type: none"> 1. You can only move down, right, or diagonally right-down at each step. 2. Each cell can only be visited once during the path. <p>Input Format:</p> <ol style="list-style-type: none"> 1. Two integers N and M representing the number of rows and columns in the grid. 2. N lines, each containing M integers $W[i][j]$, representing the weight of the grid cells. <p>Output Format:</p> <p>Output a single integer: the maximum path sum from (1,1) to (N,M).</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 1000$. • $0 \leq W[i][j] \leq 10^4$ <p>Example Input:</p> <pre>3 3 1 2 3 4 5 6 7 8 9</pre> <p>Example Output:</p> <pre>21</pre> <p>Explanation:</p> <ul style="list-style-type: none"> • One possible path: (1,1)→(2,2)→(3,3) with sum $1+5+9=15$ • The optimal path: (1,1)→(1,2)→(1,3)→(2,3)→(3,3) with sum $1+2+3+6+9=21$ <p>Code:</p> <pre>import java.util.Scanner;</pre>		
--	--	--	--

```

public class prac9 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int N = scanner.nextInt();
        int M = scanner.nextInt();

        int[][] grid = new int[N][M];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int[][] dp = new int[N][M];
        dp[0][0] = grid[0][0];

        for (int j = 1; j < M; j++) {
            dp[0][j] = dp[0][j - 1] + grid[0][j];
        }

        for (int i = 1; i < N; i++) {
            dp[i][0] = dp[i - 1][0] + grid[i][0];
        }

        for (int i = 1; i < N; i++) {
            for (int j = 1; j < M; j++) {
                dp[i][j] = grid[i][j] + Math.max(dp[i - 1][j],
                Math.max(dp[i][j - 1], dp[i - 1][j - 1]));
            }
        }

        System.out.println(dp[N - 1][M - 1]);
    }
}

```

Output:

```

PROBLEMS 9 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS
PS D:\Probin's Work\DSA\mcps> javac prac9.java
PS D:\Probin's Work\DSA\mcps> java prac9
3 3
1 2 3
4 5 6
7 8 9
29
PS D:\Probin's Work\DSA\mcps> |
Square Root Decomposition

```

10

	<p>You are given a delivery system with N locations connected by bidirectional roads. Each road has a delivery cost. Your task is to process Q queries efficiently, where each query asks for the minimum delivery cost between two locations A and B.</p> <p>You need to optimize the solution using Square Root Decomposition and Dynamic Programming.</p> <p>Input Format:</p> <ol style="list-style-type: none"> 1. N (number of locations), M (number of roads), Q (number of queries). 2. M lines, each containing three integers U, V, C representing a road between U and V with cost C. 3. Q lines, each containing two integers A, B representing a query asking for the minimum delivery cost between locations A and B. <p>Output Format:</p> <p>For each query, output a single integer: the minimum delivery cost. If no route exists, output -1.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq N, M \leq 10^5$ • $1 \leq C \leq 10^4$ • $1 \leq Q \leq 10^5$ <p>Example Input:</p> <pre> 5 6 3 1 2 4 2 3 2 3 4 6 1 5 10 4 5 1 3 5 7 1 4 </pre>		
--	---	--	--

2 5

3 1

Example Output:

12

8

-1

Explanation:

- Query 1: Minimum cost from 1→4 is 1→2→3→4 with cost 4+2+6=12.
- Query 2: Minimum cost from 2→5 is 2→3→5 with cost 2+7=8.
- Query 3: No valid path from 3→1, so output -1.

Code:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class prac10 {
    public static void main(String[] args) {
        int n = 5;
        int m = 6;
        int q = 3;
        int[][] roads = {
            {1, 2, 4},
            {2, 3, 2},
            {3, 4, 6},
            {1, 5, 10},
            {4, 5, 1},
            {3, 5, 7}
        };
        int[][] queries = {
            {1, 4},
            {2, 5},
            {3, 1}
        };
        int[][] graph = new int[n + 1][n + 1];
        for (int i = 1; i <= n; i++) {
            Arrays.fill(graph[i], Integer.MAX_VALUE);
            graph[i][i] = 0;
        }
        for (int[] road : roads) {
            graph[road[0]][road[1]] = road[2];
        }
    }
}
```

```

        graph[road[1]][road[0]] = road[2];
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (graph[i][k] != Integer.MAX_VALUE &&
graph[k][j] != Integer.MAX_VALUE) {
                    graph[i][j] = Math.min(graph[i][j], graph[i][k] +
graph[k][j]);
                }
            }
        }
    }
    for (int[] query : queries) {
        int result = graph[query[0]][query[1]];
        System.out.println(result == Integer.MAX_VALUE ? -1 :
result);
    }
}
}

```

Output:

```

PROBLEMS 13 OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS
PS D:\Probin's Work\DSA\mcps> javac prac10.java
● PS D:\Probin's Work\DSA\mcps> java prac10
11
9
6
❖ PS D:\Probin's Work\DSA\mcps> |

```